

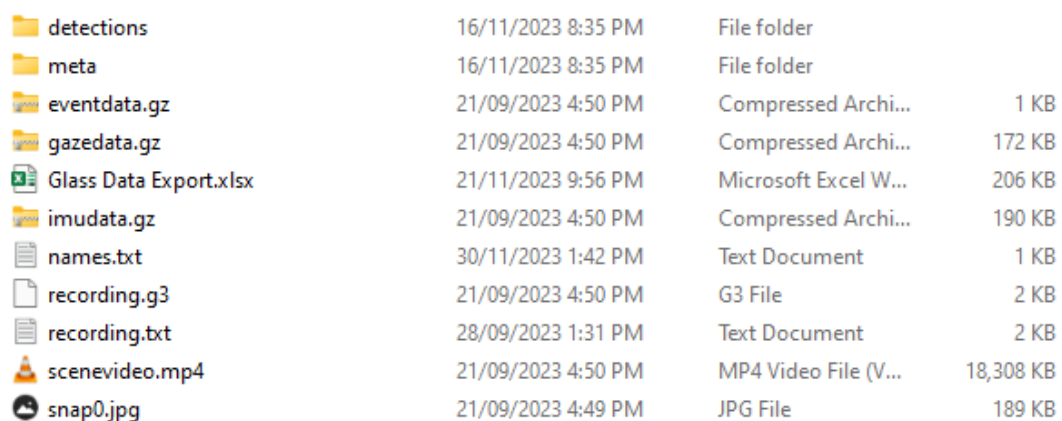
How does EyePort work?

This document provides an in-depth, detailed explanation of the major algorithms working behind the interface, including areas of interest, unique areas of interest, head orientation times, object detection, radar violation, no-go zones, and dead man switch. This can be useful in future research or to any software developer looking into the source code of EyePort for further improvements or customisation.

The content in this document is deemed beyond an average user's grasp, and so was not included in the main user manual. Please note that this software was developed for TOBII Pro Glasses 3. It will not work with any other glasses. The document only describes the algorithms behind the analysis page. Most are self-created from scratch.

Eye-Tracking Data Formats

Eye-tracking data is stored under the recorded session's folder. EyePort is only concerned with the **gazedata.gz** and **imudata.gz** in the folder. Additional checks are performed to verify that these files are present. They are stored in the form of compressed JSON (JavaScript Object Notation). The JSON files are extracted using Python libraries to fetch the glass data. EyePort converts them into an Excel file called **Glass Data Export.xlsx** for easier access. Appropriate writing permissions must be present for the folder (Read-only or Windows Controlled Folder Access will cause an error).



detections	16/11/2023 8:35 PM	File folder	
meta	16/11/2023 8:35 PM	File folder	
eventdata.gz	21/09/2023 4:50 PM	Compressed Archi...	1 KB
gazedata.gz	21/09/2023 4:50 PM	Compressed Archi...	172 KB
Glass Data Export.xlsx	21/11/2023 9:56 PM	Microsoft Excel W...	206 KB
imudata.gz	21/09/2023 4:50 PM	Compressed Archi...	190 KB
names.txt	30/11/2023 1:42 PM	Text Document	1 KB
recording.g3	21/09/2023 4:50 PM	G3 File	2 KB
recording.txt	28/09/2023 1:31 PM	Text Document	2 KB
scenevideo.mp4	21/09/2023 4:50 PM	MP4 Video File (V...	18,308 KB
snap0.jpg	21/09/2023 4:49 PM	JPG File	189 KB

Figure 1 – Record Folder Structure

Additionally, the **scenevideo.mp4** is the actual recording from the front camera of the glasses. The resolution of this video is 1920 × 1080 recorded at 25 frames per second. This file is needed for the Playback Data page and still frames for processing Object detection and Areas of Interest display sections under the Analyse Data page.

There is an additional bug if the glasses are used in Video Mode or Full Analysis Mode in

TOBII's own recording software. Switching between the modes causes a bit in the **recording.g3** file to flip, causing no gaze circle to be overlaid in the scenevideo.mp4. EyePort checks for the bit in the recording.g3 file, ensuring that gaze circles are always present for displaying under the Analysis Page section. It will add its own circles from the extracted Excel file data onto the still frames if the gaze circles are found to be absent. This is a fix only for the Analysis Page but not for the Playback Page.

Overall, four files are needed for complete analysis. Should these files become inaccessible at any point, EyePort will raise an error. EyePort creates additional files in the folder, that takes some time, but the process is only done once. EyePort will only do the extraction if the extracted files are not present. This prevents re-extraction and saves resources on the user's computer.

Areas of Interest Detection

Most algorithms, including detecting areas of interest, occur before extracting frames from the video and displaying them on EyePort. Hence, there is a delay after the "Analyze Now" button is clicked. They are also located in the `Data_Analysis_Rotating_Head.py` file.

To get started, the entire extracted Excel file **Glass Data Export.xlsx** is analysed in a loop. This is important information because if the user's computer has fewer CPU cores, this process will take longer. EyePort must be run on at least Octa Core systems. EyePort looks for intermediate changes in the Gaze 2D coordinates over a fixation time set by the user. These are the coordinates of where the eyes are fixed on the 1920×1080 scenevideo. The values are a pair of decimal values from 0 to 1 in the glass data file. Implicit conversion is done from this decimal to the pixel coordinate of the video.

For example, if the Gaze 2D data is 0.5, 0.5. This signifies a pixel coordinate of $1920 \times 0.5 = 960$ in the x-axis and $1080 \times 0.5 = 540$ in the y-axis. Therefore, the user is looking at the centre of the video (or vision).

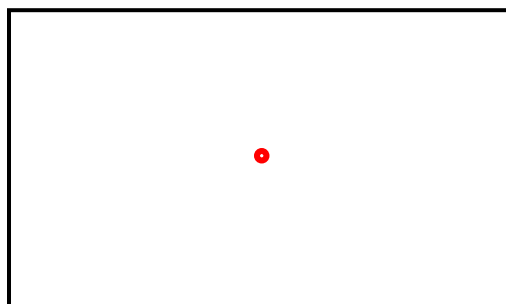


Figure 2 – Centre Vision Example

A larger **tolerance rectangle** of size 96×54 pixels is always calculated for the next expected gaze coordinate. This is an excellent time to mention that the density of data in the glass data file is 100 data points per second (100Hz). Therefore, calculations can be as accurate as

1/100th of a second or 0.01 seconds. But to prevent instability and incorrect logic errors, the fixation time is restricted to only 1/4th of a second or 0.25 seconds.

If the user's gaze stays within the tolerance rectangle for more than 25 data points, which means a fixation time of 0.25s, the timestamp is recorded for the starting time. Here is an overview of the process.

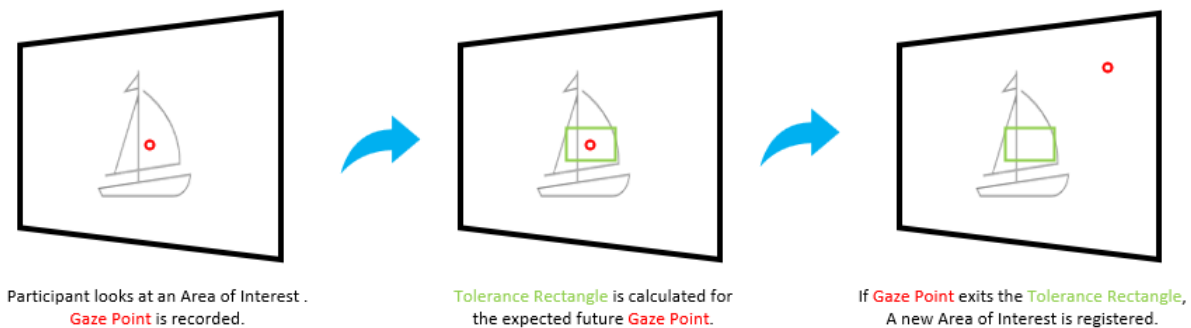


Figure 3 – Areas of Interest Detection

Fixation Time of 0.5 Second = 50 Data Points

Fixation Time of 2.5 Seconds = 250 Data Points, and so on.

Whenever the gaze point exits the tolerance rectangle, one of the following can happen:

- There were sufficient data points more than the fixation time to signify that the user had looked at the object long enough to register it as an Area of Interest. The end timestamp is recorded.
- There were not enough data points to exceed the fixation time. No timestamps are recorded. This case is discarded, and the counters are reset. For example, blinks, eye saccade movements, etc.

There were mentions of start and end timestamps before. These are then displayed in the tables under the analysis page. These data also help create graphs, calculate the duration of how long the area of interest was looked at, determine which frames to extract from the video to show the user, and so on.

Note that the user **cannot** change the tolerance rectangle dimensions as other dimensions of this rectangle were tested and significantly affected the algorithm's performance. The only control the user has is the ability to adjust the fixation time for this section. The algorithm developed for interest areas has proven reliable and flawless in most test cases, including tracking slow-moving objects. There were other ways of detecting the areas of interest in the previous work term. Complex matrix calculations, single integrations, and vector tolerance matching (not discussed in this document) were done in earlier versions of EyePort for TOBII Pro Glasses 2. The reason this algorithm was replaced completely was because of the following:

- More Calculations (Potentially slowing down user systems)
- Higher Chances of Error (If gyroscope calibration was not done correctly)
- New Glasses had a worse Gyroscope (values starting in the 7-10 degrees/second range even if the glasses are stationary)
- Difficulty Filtering Errors and Calibrating the Gyroscope

Although the new algorithm is better, it has the potential downside of relying only on user gaze data to register areas of interest. If the user quickly turns their head whilst still looking at the same place in the vision, the new area is not detected (an unlikely scenario, but it is possible). This vulnerability was found in the full motion bridge simulator trials that took place in November 2023 at the Marine Institute.



Figure 4 – Full Motion Bridge Simulator Trials

Post Synchronization Step

Previously, it was mentioned that the glass data samples were at 100Hz (100 data points per second). However, the **scenevideo.mp4** file only contains 25 frames per second, restricting accuracy to only 1/25th of a second or 0.04 seconds. Because the number of glass data points is always greater than the available frames, an additional post-synchronization must be done. This is a “lossless synchronization”, meaning that no data is ever lost in syncing the two data: video frames and glass data. This step always takes right before the EyePort interface is refreshed and only after all algorithms (other than object detection) have finished processing. This ensures that algorithms have the most data to work with (video frames are not required at this stage).

The glass data and video frames have their own timestamps. The timestamps of the glass data have 7-8 significant digits (Example – 1.234567 seconds is 1234567). These numbers do not follow any specific timing interval but are roughly 10ms apart, which matches the 100Hz sampling rate of the glasses. One can verify this with the **Glass Data Export.xlsx** file created

by EyePort. But video timestamps are different and do follow a timing interval of 0.04 seconds for 25 fps. Conversion is needed from glass to video timestamps. The two timestamps are matched by taking their overall length and ratios. This will spread the video frames over the glass data points, as shown in Figure 5. Any non-existent frames needed are automatically matched to the nearest frame available.

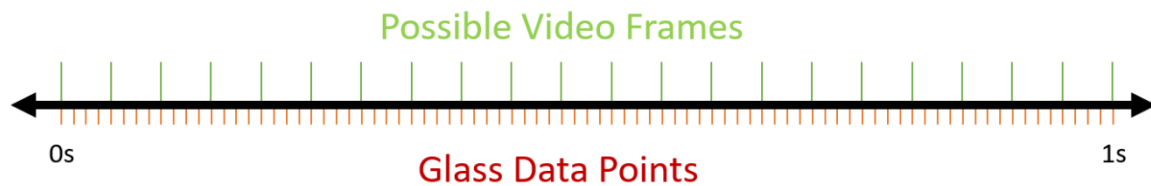


Figure 5 – Video Frames vs Glass Data Points

For example, if a glass data point is identified to be an Area of Interest at 25.067 seconds (video timestamp and not the glass timestamp) after the post-synchronization step. The two closest frames that can be used here are at 25.040 and 25.080 seconds. EyePort will select the frame at 25.080 seconds as it is the closest to 25.067 seconds. This way, all Areas of Interest are guaranteed to have a frame even if it did not exist at the actual second of the video.

Green Frame Bug: During development, it was found that sometimes the **scenevideo.mp4** is not rendered correctly in the first few frames of the video. They were found to be just green still frames, indicating an encoder failure at the glass hardware level. Therefore, if an Area of Interest is registered within the first few frames of the video (<0.1s), it would have a green still frame instead of the actual vision frame. To overcome this, the first 3 frames are ignored for processing. Instead, the 4th frame is chosen for the frame image.

Unique Areas of Interest Detection

All detections of areas of interest are stored in a folder called detections under the main record folder which are cropped into a smaller image (square dimensions defined by the user) from the entire 1920 × 1080 frame. This is used for Unique Areas of Interest Detection. Imagine a square box around the gaze point. Notice the square images formed in Figure 6.



Setting a HIGH side length will increase the number of Unique AOIs. However, this may falsely mark the same object as different objects.

Setting a LOW side length radius will decrease the number of Unique AOIs. However, this may falsely mark different objects in close proximity as the same object.



Figure 6 – Sample contents of detections folder

Using a Python library called `image_similarity_measures`, the areas of interest are matched against each other. Similar-looking images are grouped as one object, and the rest are left as is. Sequential numbering is taken from Object 1, Object 2, and so on. The detection is based on how similar the images are in structure and RGB colour information. The user can control the sensitivity of this matching under the Detection Sensitivity control.

```
def updateasense(a: str):
    global asense
    if a=='Extremely Low':
        asense=0.40
    elif a=='Very Low':
        asense=0.55
    elif a=='Low':
        asense=0.65
    elif a=='Normal':
        asense=0.75
    elif a=='High':
        asense=0.85
    else:
        asense=0.95
```

Figure 7 – Code for selecting the sensitivity.

The higher the sensitivity (Figure 7), the matching tolerance increases. The maximum value is 0.95, meaning the two images need to have a similarity of 95%. On the other hand, if 0.40 is selected, the two images need to have a similarity of 40%.

Manual Mode completely bypasses this algorithm, allowing the user to choose which objects are the same, giving full control to the user and ensuring a fully correct human-verified analysis page. This mode marks all areas of interest as unique and modifies the “Edit Names” button to define the unique areas of interest manually.

Based on Figure 6, it is clear that Object 1,4 is the same while Object 2 is different. Object 3 may or may not be the same as Objects 1 and 4, but this can be adjusted with the sensitivity control or using manual mode altogether.

Blink Detection

Unfortunately, EyePort does not support blink detection. When these events occur, the glass data in **Glass Data Export.xlsx** are not present. Because the extraction algorithm is designed only to create rows of data that include the **full range of data** from various sensors (that includes Gaze 2D coordinates). If any of the sensors cannot provide data for a specific timestamp, that row is skipped and not created in the Excel file. This means that blinks are ignored as the Gaze 2D Coordinates are empty during that time. EyePort also raises an error if the participant never wears the glasses for this very reason. Data is omitted this way because most algorithms require a steady stream of data points. Additional checking codes for such cases could slow down analysis times or create complications during post-synchronization or frame extraction (needing gaze points).

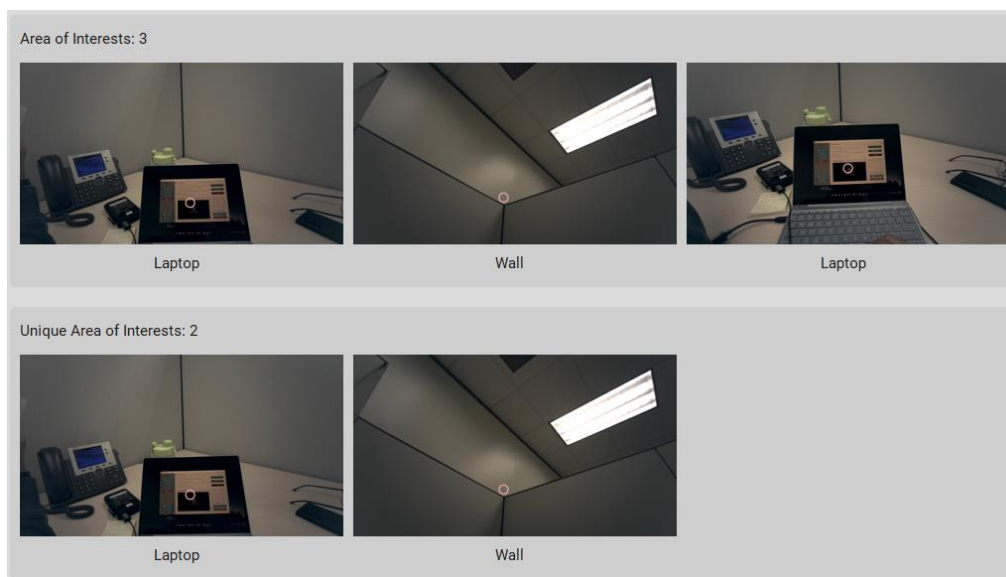


Figure 8 – Sample Detection of Laptop and Wall

Head Orientation Times

This algorithm depends on the gyroscope values in the y-axis from the extracted glass data and two values from the user. This is also a loop-based algorithm.

```
#Calculating Head Up, Head Level, Head Down Times
headup,headlevel,headdown = [],[[0,]],[],[]
state = headmode
if state==2: #From Level Start
    headup,headlevel,headdown = [],[[0,]],[],[]
elif state==3: #From Down Start
    headup,headlevel,headdown = [],[],[[0,]],[]
else: #From Up Start
    headup,headlevel,headdown = [[0,]],[],[],[]
ready = True
starter = False
#heads User defined acceleration value
for i in range(len(GX)): ...
```

Figure 9 – Code Snippet for Head Orientation Times

Figure 9 shows a small snippet of the code. The `headup`, `headlevel`, and `headdown` lists contain the start and timings of each head orientation. This is where the user **needs to specify** how the participant's head was oriented at the start of the recording. This is crucial for the calculations within the loop (code not shown) and because gyroscope calibration was deemed difficult for EyePort, as discussed in the Areas in Interest section. Notice how the starting 0 seconds differs for each starting orientation in Figure 9.

For example, if the lists contain values in the following format:

```
headup = [[2,4],[6,9]]
```

```
headlevel = [[0,2],[4,6]]
```

```
headdown = [[9,11]]
```

This means that the participant's head was level at the start of the recording for 2 seconds. Then there was a head-up state from 2 to 4 seconds, then head level for 4 to 6 seconds, head up again for 6 to 9 seconds, and finally head down for 9 to 11 seconds. If the participant started from the head-up orientation for 4 seconds, then the list should be `headup = [[0,4]]`.

The **second information** EyePort needs from the user is the degrees per second to detect the “jolt” during which the head orientation changes. This can be defined by the user in EyePort settings. Each time a participant moves their head with sufficient angular speed

more than the user-defined degrees/second, the corresponding lists will be altered with start and end times based on the head state.

To avoid false readings during the transition phase, the timers responsible for counting the head up/level/down times are **temporarily paused**. They are resumed when the head reaches an "expected" next state, and the gyroscope readings are settled close to 0 degrees/second. There is an estimated loss of 2 seconds between head movements. So the timed lists shown earlier would realistically have the following format:

```
headup = [[2.82, 3.53], [6.23, 8.66]]
```

```
headlevel = [[0, 1.53], [4.45, 5.10]]
```

```
headdown = [[9.23, 10.35]]
```

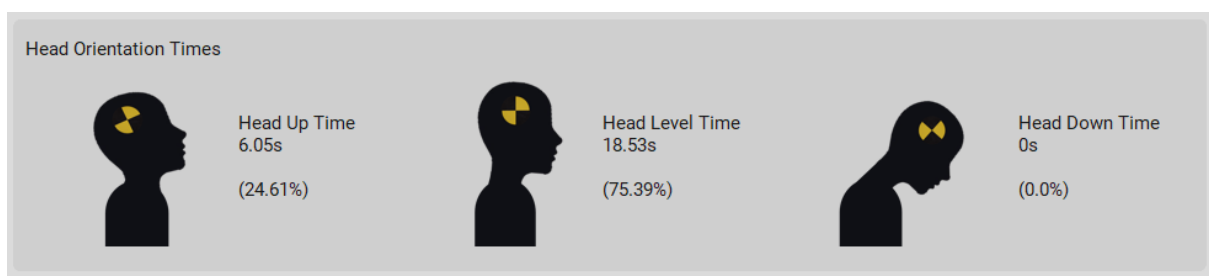


Figure 10 – Sample Head Orientation Times

The percentages and total durations in Figure 10 (not the same as the timed list example) are calculated separately in another algorithm, taking their information solely from the three critical timed lists.

Object Detection

A major part of this algorithm's functionality comes from the ImageAI open-source repository found at <https://github.com/OlafenwaMoses/ImageAI>. Additional helper codes were integrated into EyePort to use object detection. This library essentially allows the computer to automatically detect the objects being looked at and rename them before displaying them in the EyePort interface while not requiring internet connectivity. This is achieved using pre-trained models like a neural network to classify the objects.

The library, along with the accompanying models for General Objects, Ships and Icebergs, and VISTA Diesel Engine, is what takes 90% of EyePort's installation size (1.5 GB). Therefore, since Version 3.2.2, a lite release of EyePort was built to add these features as an optional add-on keeping installation sizes to a minimum.

All **Unique Areas of Interest** are passed into this algorithm, which returns a list of "answers" containing the detection results. Using detection takes significantly longer, as multiple frames are being cropped into tiny squares to compare with the trained model. This requires intense CPU and GPU performance. A brief overview of the process is shown in Figure 11.

Creating a pre-trained model involved the collection of over 500 images of ships and icebergs and screenshots of the VISTA Diesel Engine Simulator that had to be annotated manually in an annotation format called YOLO. This annotation contains information about the object and the coordinates of the object's bounding box in the image. A tool called Labellmg was used to mark the bounding boxes around objects and label each object in the image manually. Additional information can be found in the author's [documentation](#).

The training process took about two days with 500 passes. Any objects not recognised are shown as "Detection Failure" in EyePort.

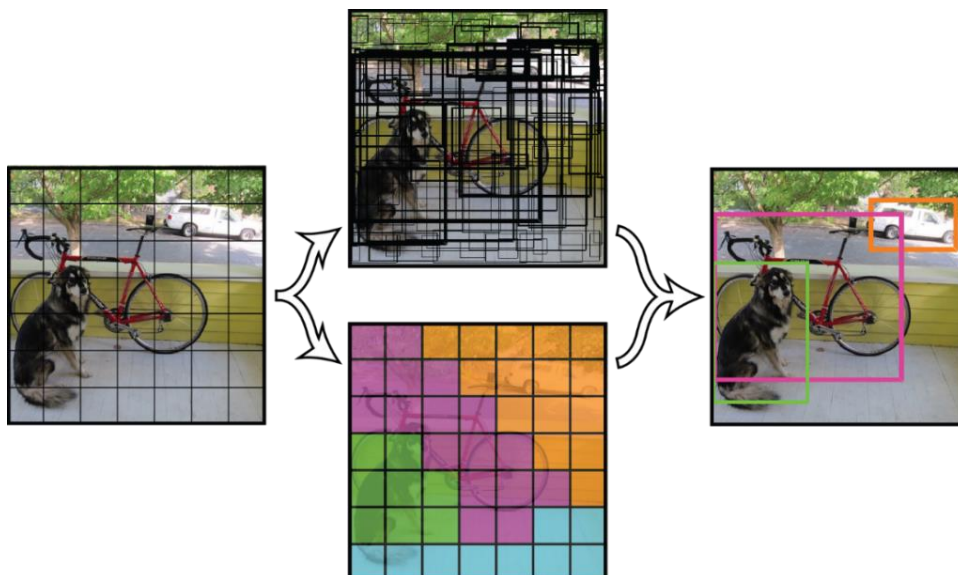


Figure 11 – Object Detection Procedure

No-Go Zones

This algorithm is very similar to the unique areas of interest detection algorithm. It uses the same `image_similarity_measures` library, but this time, the similarity matching tolerance is increased to 0.95 (95% matching needed).

Also, the matching is done with pre-defined images located in `EyePort\NoZones` instead of Areas of Interest. Users can open this location using the “Define No Go Zones” button under Analysis Preferences. To define No-Go Zones, please refer to the user manual.

Define No Go Zones

Any objects matching this criterion will be displayed as a separate frame under the No-Go Zones section.

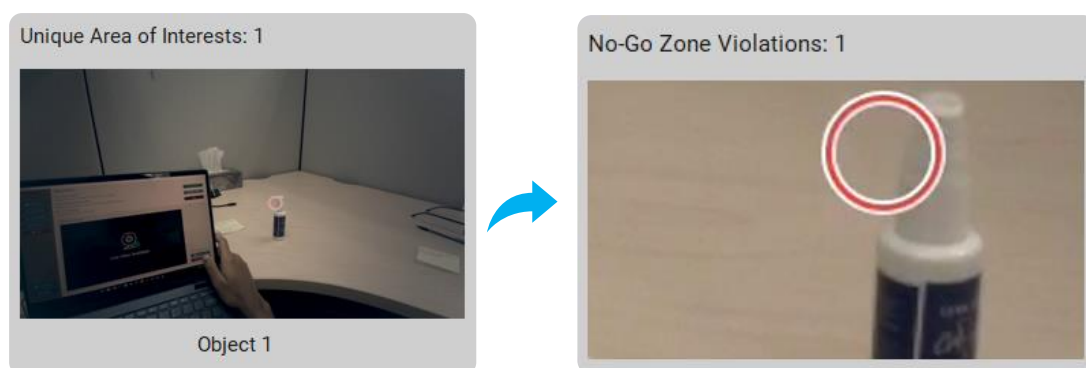


Figure 12 – Sample No-Go Zones

Radar Violation and Dead Man Switch

These algorithms are smaller compared to the others discussed in this document. They rely heavily on the information incoming from the other algorithms.

The radar violation algorithm checks for the “Radar” area of interest in the list of interests and keeps track of the times coming from the Start and End Times calculated in the Areas of Interest detection algorithm.

```

def radar():
    global aviolate
    flag = False
    check = [0,]
    violation = []
    violation_type = []
    for i in range(len(AOIName)):
        if AOIName[i]=="Radar" and (StartSec[i]-check[-1])<aviolate:
            check.append(StartSec[i])
        elif AOIName[i]=="Radar" and (StartSec[i]-check[-1])>aviolate:
            violation.append(StartSec[i])
            violation_type.append("Looked at Radar too late")
            check.append(StartSec[i])
        elif AOIName[i]!="Radar" and (StartSec[i]-check[-1])>aviolate:
            violation_type.append("Failed to look at Radar")
            violation.append(StartSec[i])
            check.append(StartSec[i])
        else:
            pass

    if len(violation)==0:
        tk.messagebox.showinfo(lang[126],lang[127]) #No Radar Violations
    else:
        violations = "\n"
        for i in range(len(violation)):...
        tk.messagebox.showwarning(lang[128],f"{lang[129]}{violations}")

```

Figure 13 – Radar Violation Detection Code

The user can change the interval duration for this algorithm, which is stored in `aviolate` variable (Figure 13). The rest of the checks mentioned earlier are done in a small loop. If no violations are found, an information message is shown. Otherwise, an error message is generated.

Lastly, the dead man switch checks the duration of the head orientation states and the areas of interest. If they are **both** too long (>45 seconds), the dead man switch will trigger. The code for this is only a few lines, as shown in Figure 14.

```

flag1,flag2 = False,False
for i in range(len(AOIName)):
    if (round(EndSec[i]-StartSec[i],2)>45):
        flag1 = True
        break

for i in headduration:
    if (i>45):
        flag2 = True
        break

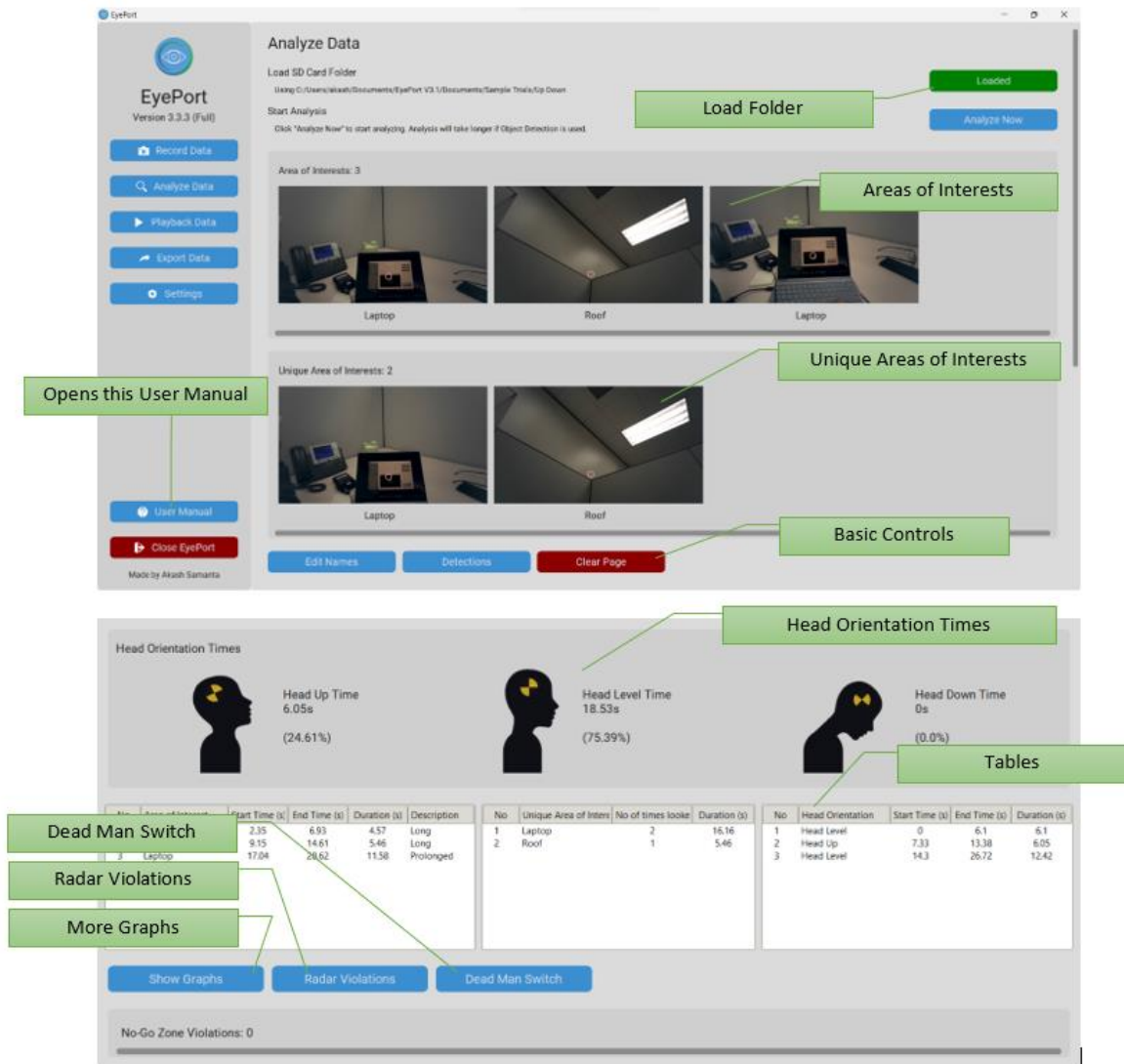
if flag1 and flag2:
    deadmantext.configure(text=lang[110]) #Dead Man Switch: Active

```

Figure 14 – Dead Man Switch Code

What happens next?

This is what happens when the user clicks the “Analyze Now” button behind the scenes. After all the discussed algorithms are complete, the interface of EyePort is finally updated to show the result. The playback page is also enabled and will be ready for playback video with overlays. If, at any point, an unforeseen error occurs while executing all the algorithms, the “Failed to Analyze” Error (Figure 15) will be generated.



Or..

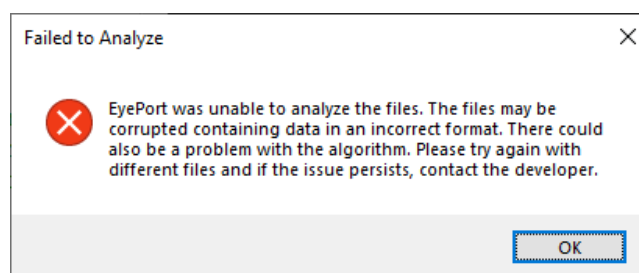


Figure 15 – Final Result